Achieve Fuzzing Based on Symbolic Execution Platform

He Hao[1, a *] and Gan Shuitao[2,b]

[1] Lab of Computer Science, Jiang Nan Computer Technique Institute, Shanshui Dong Road Wu Xi, Jiangsu 214000, China

[2] Lab of Computer Science, Jiang Nan Computer Technique Institute, Shanshui Dong Road Wu Xi, Jiangsu 214000, China

[a]342118567@qq.com, [b]runteemo@foxmail.com

Keywords: fuzzing, symbolic execution

Abstract. The traditional software development does not consider the concerns, which is can addressing security problem effectively. So how to find the problem is the standard consideration for security researchers. In the field of software security testing, there are four ways, Fuzzing, Dynamic Taint Analysis, Model Checking and Symbolic Execution. This paper we use symbolic execution to achieve functional of fuzzing.

Introduction

With the development of computer technology, there are more and more software applied in every walk of life, so the security of software have been known by developer. However, we cannot avoid the exception in developing software, the method of software vulnerability analysis is an important way to protect the security of information system.

There are four ways in this field.

Dynamic taint analysis [1] is belongs to data-path analysis. It sign the data with untrusted source to the taint, and tracking the data. Model checking [2] will model a system and uses formal method to judge the system whether it is a satisfied system properties. Fuzzing [3,4] can provide random input data to find the software vulnerability. Symbolic execution [5] is also belongs to data-path analysis, it use symbolic input instead of actual input, so it can generate more targeted test cases automatically.

Symbolic Execution

Theory. The program variable is a symbolic representation in symbolic execution, and because it is a data-path analysis method, in the process of analysis, it collects path constraint conditions with instrumentation, and generate test cases to find the software vulnerability.

The example of symbolic execution is shown in the figure below.

```
1.  void example (int x, int y){
2.      if ( x < 5 )
3.          assert ( x < 5 );
4.      else if ( y < 3 )
5.          Assert ( x< 5 && y < 3 );
6.      else assert ( x > 5 && y > 3 );
7.          end;
8.  }
```

Fig.1 This is an example of symbolic execution.

In this example, the symbolic execution is represented as follows:

value : (0, 0) path : 1-2-3-7-8 path_condition (x < 5).

The new path condition is !(x < 5), so we can get x = 5.

value : (5, 0) path : 1-2-4-5-7-8 path_condition !(x < 5)&&(y < 3).

The new path condition is !(x < 5)&&(y < 3), so we can get x = 5, y = 3.
value : (5, 3) path : 1-2-4-6-7-8 path_condition !(x < 5)&&!(y < 3).

In this way, when one symbolic execution process finished, we can use constraint solver to get a set of specific input. And when all processes over, the symbolic execution finished.

S2E

Application. S2E means selective symbolic execution. The core concept is doing symbolic execution for object code which we will focus on, such as library function. And for the system kernel function and other application function it doing concrete execution. The schematic diagram of S2E [6] is shown in the figure below.

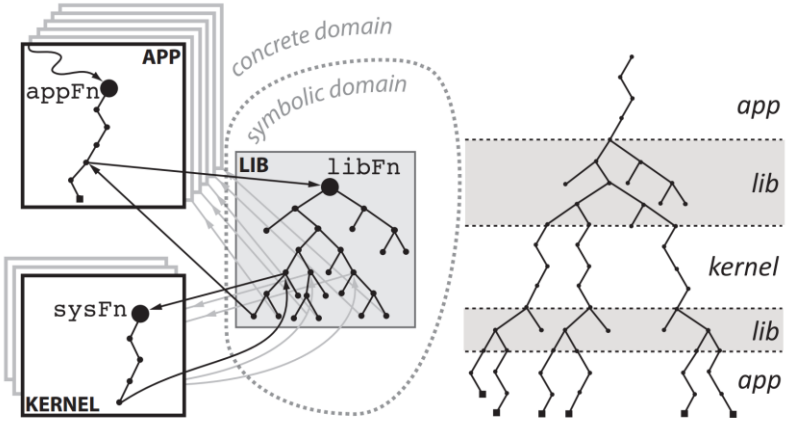

Fig.2 This is a schematic diagram of S2E.

Framework. The S2E is based on QEMU, KLEE [7] and LLVM. In this platform, LLVM translating the C code to BC code. KLEE doing the symbolic execution. QEMU running the target program. Also, there are some selectors and analyzers in the platform, these plugins have good expansibility.

Improvement

Code. We can write codes in exception interceptor plugin. First we need add register handler in this plugin. Second we need add the handle for the ZERO. Finally, we need add the event in the exceptions. These codes are shown in the figure below.

```
1. registerHandler (state, ZERO);
```

```
1. case ZERO:
2.     s2e()->getMessagesStream() << “Find zero fault\n”
3.     break;
```

```
1. enum EX86Exceptions{
2. ZERO = 0;
3.  …
4. }
```

Fig.3 This is the codes of Divide by zero.

Experiment. We use the improved platform to test an example test.c, and catch the exception successfully.

Summary

We can use the same way to improve the plugins, and in view of the specific execution exception events, S2E can have testing methods. According to improve S2E, this platform can implement functions of fuzzing.

References

[1] W. Halfond, A. Orso, and P. Manolios. Using Positive Tainting and Syntax aware Evaluation to Counter SQL Injection Attacks[C]. In Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, New York, USA, 2006.
[2] A. Tsitovich. Detection of Security Vulnerabilities Using Guided Model Checking[C]. In ICLP'08: Proceedings of the 24th International Conference on Logic Programming. Udine, Italy, 2008.
[3] MILLER B P, KOSKI D, LEE C P, et al. Fuzzing revisted: a reexamination of the reliability of UNIX utilities and services[R].Madison: University of Wisconsin Madison, 1995.

[4] SUTTON M, GREENE A, AMINI P. Fuzzing: brute vulnerability discovery [M].[S.l.]: Pearson Education Inc, 2007: 16.

[5] J.C. King. Symbolic execution and program testing [J]. Communnications of ACM, 1976, 19(7): 385 394.
[6] V Chipounov. S2E: A platform for in-vivo multi-path analysis of software systems [C]. AS PLOS '11 Volume 46 Issue 3, March 2011.

[7] C. Cadar, D. Dunbar, D. R. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs[C]. In Symp. on Operating Systems Design and Implementation, 2008.